AD-A069 458 ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE F/G 9/2

VTMS: A VIRTUAL TERMINAL MANAGEMENT SYSTEM FOR RIG.(U)

MAY 79 K A LANTZ, R F RASHID N00014-78-C-0164

UNCLASSIFIED TR-44 NL

| OF |

AD

A069458

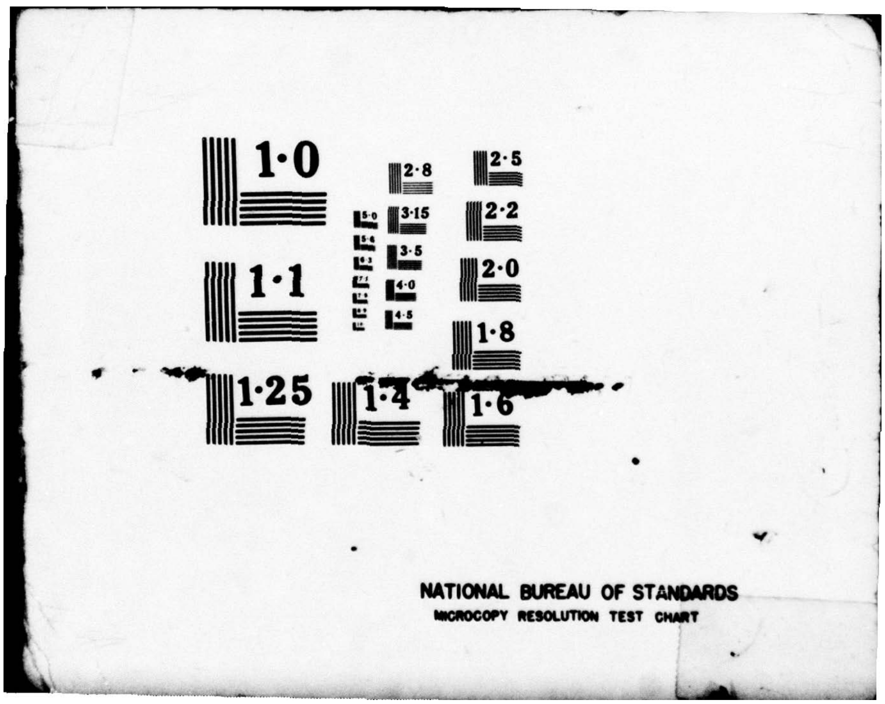
END

DATE

FILMED

7-79

DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.15
3.5
4.0
4.5
5.0
5.6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

電腦科學

rochester

Department of Computer Science
University of Rochester
Rochester, New York 14627

VTMS:
A Virtual Terminal Management System
for RIG

Keith A. Lantz and Richard F. Rashid
Computer Science Department
University of Rochester
Rochester, NY 14627

TR44
May 1979

Abstract

Rochester's Intelligent Gateway provides its users with the facilities for communicating simultaneously with a large number of processes spread out among various computer systems. We have adopted the philosophy that the user should be able to manage any number of concurrent tasks or jobs, viewing their output on his display device as he desires. To achieve this goal the Virtual Terminal Management System (VTMS) converts a single physical terminal into multiple virtual terminals, each of which may be written into or queried for user input. VTMS extends the features of the physical terminal by providing extensive editing facilities, the capacity to maintain all output in disk-based data structures, and sophisticated mechanisms for the management of screen space. Virtual terminals are device-independent; the specific characteristics of the terminal at hand are known only to the lowest-level I/O handlers for that device. VTMS is currently running on a network of six minicomputers supporting various text and raster-graphics displays.

The research described in this report was supported in part by DARPA grant N00014-78-C-0164 and NSF grant MCS76-10825.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR44	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): 6 VTMS: A Virtual Terminal Management System for RIG	5. TYPE OF REPORT & PERIOD COVERED: 9 technical report
	6. PERFORMING ORG. REPORT NUMBER: 44
7. AUTHOR(s): 10 Keith A. Lantz ~~and~~ Richard F. Rashid	8. CONTRACT OR GRANT NUMBER(s): 15 N00014-78-C-0164, new NSF-MCS76-10825
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, University of Rochester, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Voulevard, Arlington, VA 22209	12. REPORT DATE: 11 May 1979
	13. NUMBER OF PAGES: 39
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217 — 12 41p.	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

14 TR-44

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

—

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

concurrent operations	network access
device-independence	terminals
distributed computing	user interfaces
multiprocessing	virtual terminals

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Rochester's Intelligent Gateway provides its users with the facilities for communicating simultaneously with a large number of processes spread out among various computer systems. We have adopted the philosophy that the user should be able to manage any number of concurrent tasks or jobs, viewing their output on his display device as he desires. To achieve this goal a single physical terminal is converted in multiple virtual terminals, each of which may be written into or queried for user input. Virtual terminals are device-independent and provide extensive editing facilities and the capacity to maintain all output on disk.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 386 Gu

CONTENTS

1.0 Introduction

Rochester's Intelligent Gateway (RIG) provides its users with the facilities for communicating simultaneously with a large number of processes spread out among various computer systems [Ball, et al., 1976; Ball, et al., in prep.]. Our six in-house minicomputers and local PDP-10 provide editing, file transfer, printing, program compilation and execution and other services which may be accessed directly or through the auspices of special interface processes. In addition, we link into the ARPANET as a Very Distant Host; Telnet and File Transfer tools capable of providing concurrent communication with several remote computers are available [Roberts and Wessler, 1971].

Given an environment in which so much simultaneous activity is possible, one fundamental problem is how to translate that activity into a form comprehensible to the user. We have adopted the philosophy that the user should be able to simultaneously manage any number of concurrent tasks or jobs,* viewing their output on his display device as he desires. This is based on the belief, shared by Teitelman, that:

> Being able to switch back and forth between tasks results in a relaxed and easy style of operating more similar to the way people tend to work in the absence of restrictions. To use a programming metaphor, people operate somewhat like a collection of coroutines corresponding to tasks in various states of completion. These coroutines are continually being activated by internally and externally generated interrupts, and then suspended when higher priority interrupts arrive, e.g., a phone call that interrupts a meeting, a quick question by a colleague that interrupts a phone call, etc. ...it is of great value to the user to be able to switch back and forth quickly between related tasks. [Teitelman, 1977]

Moreover, the command interaction discipline should be consistent across all tasks; this includes the use of control keys, invocation of help facilities, prompting, feedback, etc. The implementation of these beliefs results in a consistent and robust interface between RIG and the user. This paper discusses the RIG Virtual Terminal Management System (VTMS), which provides the necessary terminal support functions.**

----

* In RIG, jobs are effectively a collection of processes which perform a particular task -- e.g., edit or file transfer. For the sake of consistency, the term 'process' will normally be used wherever any of the terms process, job, or task would be appropriate.

** The user interface as a whole is one of the subjects dealt with in a thesis in progress [Lantz, in prep.].

2.0 Overview

VTMS converts a single physical terminal into multiple Virtual Terminals. Each Virtual Terminal may be associated with a different process, and more than one Virtual Terminal may be mapped to the physical terminal simultaneously. Each Virtual Terminal may be written into or queried for user input just as its physical counterpart might be used in a single-job-per-terminal system. In addition, VTMS provides extensive editing facilities, the capability of maintaining all output in disk-based data structures, and sophisticated mechanisms for the management of screen space.

The design of VTMS was founded on three 'laws' of terminal management:

1. The user must have complete, preemptive control of his terminal at all times. He should be able to allocate and arrange the space on his display device at will, selecting which processes to 'view' at any one time. He should be able to tailor the system to his own preferences and needs.

2. Processes should never depend on the actual mapping of their output onto the user's display. Processes may stipulate preferred viewing conditions, but these are applicable only as long as they do not interfere with Law 1.

3. The output of any process should never be thrown away (during its lifetime) by the system unless specifically requested by the user. The user should at any time be able to examine the past activity of his processes, possibly forming new input from data displayed on the screen. He must be able to save the output of any process as a transcript file.

Stated simply, the user should be protected from the I/O whims of processes and processes should be independent of changing user I/O requirements.

Because VTMS serves as a buffer between users and processes, it is appropriate to examine the 'appearance' of VTMS from both points of view.*

----

* To avoid later confusion it may be desirable at this time to glance at the Glossary of terms provided in the Appendix. Whenever these terms are used as defined in the Glossary, they will be capitalized.

2.1 The User View

Sitting in front of his physical Terminal the user sees a collection of Virtual Terminals mapped onto rectangular areas of his Display. Each Virtual Terminal may be thought of as roughly equivalent to an independent physical display device. Only one Virtual Terminal is termed active -- the one accepting input from the user's Keyboard. Through the use of special keys and Monitor commands (see Section 7) the user is able to switch his attention from one Virtual Terminal to another.

The user may choose to see as little or as much of a particular Virtual Terminal as he wishes. He may block or discard output, or abort or suspend the process associated with a Virtual Terminal. He may at any time scroll back and forth to review previous material. He may specify a file or select previous text as the current input.

2.2 The Process View

From the perspective of a RIG process, the central component of VTMS is the Virtual Terminal. A Virtual Terminal consists of three logical components, each managed by separate processes which communicate with each other and with user processes via messages [Ball, et al., 1976]:

1. Line - A Line serves as the Virtual Terminal's source of keyboard input. It is managed by a Line Handler.

2. Pad - A Pad is a disk-based data structure used for storing and editing Virtual Terminal output. It is maintained by a Pad Handler.

3. Window - A Window is a potential mapping of a Virtual Terminal onto a Display. It is managed by a Screen Handler.

Processes deal with a single Virtual Terminal data structure which allows them to disregard the various components of the VTC. Special interface routines distribute the necessary commands to the appropriate process.

2.3 The Virtual Terminal Controller

Each RIG Terminal has its own Terminal Input and Output Handlers, created at system generation time. When a user first accesses RIG, a Screen Handler and a Line Handler are created. The Screen Handler is responsible for managing Screen space. The Line Handler is given control of the Terminal Input Handler, and is responsible for handling all subsequent input requirement for the user. Typically, one Pad Handler is associated with each process initiated by the user, and satisfies that process' output

requirements by sending display commands to the Terminal Output Handler. Taken together these processes comprise the Virtual Terminal Controller (VTC) for the user at hand. Figure 1 diagrams the flow of information within a VTC.

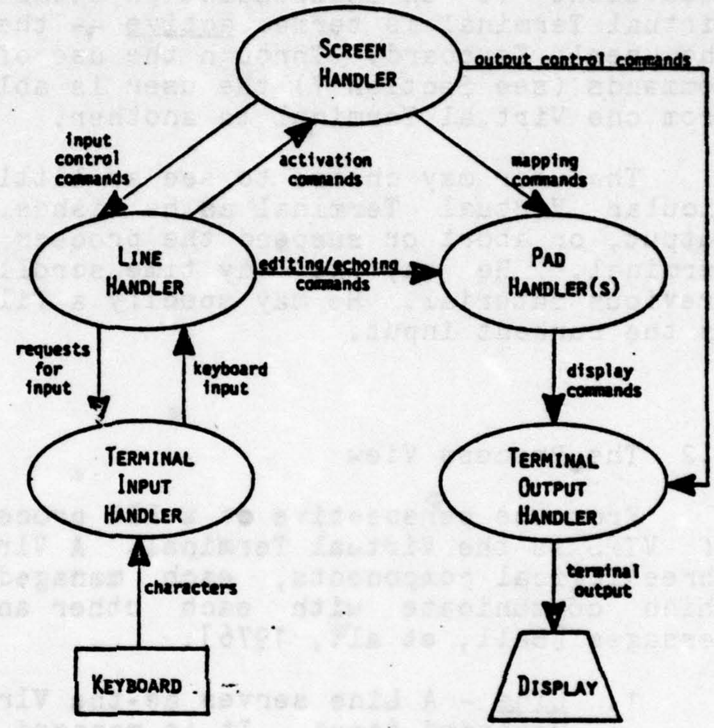


Figure 1. A Virtual Terminal Controller consists of five or more separate processes. The Terminal Input Handler collects input characters from the Keyboard, and passes them on to the Line Handler for 'interpretation.' To echo a character the Line Handler passes it to the appropriate Pad Handler, which stores the character in a Pad and passes it to the Terminal Output Handler for echoing on the Display. Editing keys result in appropriate commands being sent to a Pad Handler, which performs the edit on a Pad and issues the appropriate display updates to the Terminal Output Handler. If a character is typed which causes a new Virtual Terminal to be activated, the Line Handler notifies the Screen Handler; the Screen Handler in turn notifies the appropriate Pad Handlers to start/stop mapping their output to the Display, and tells the Terminal Output Handler which Pad is in control of the input cursor. Not shown are user process requests to the Line Handler for input, to Pad Handlers for output, and to the Screen Handler for formatting.

The division of effort within a VTC permits the standardization of input/output protocols and the distribution of components. RIG processes deal uniformly with Screen, Line, and Pad Handlers, which communicate in turn with device-specific Terminal Input and Output Handlers. These protocols are identical for all physical terminals, and are collectively referred to as the Virtual Terminal Protocol (VTP). The specific characteristics of the Terminal at hand are isolated to the Terminal Input and Output Handlers for that Terminal. Thus, the protocol between VTMS and the Terminal may vary with the Terminal, but the protocol between VTMS and the rest of RIG remains constant. This leads to reliability, flexibility, and maintainability across a potentially wide range of devices.

2.4 Outline

Sections 3 through 6 discuss implementation details. Section 3 discusses input, Section 4 output, Section 5 screen management, and Section 6 real (physical) terminals. These details are, in general, of little interest to the typical user.

The reader interested in seeing how VTMS would "look" to him as a user might find Section 7 sufficient. It presents an extended example from both the user and process point of view.

3.0 Virtual Input

Multiplexing the user requires that input and output must be shared between processes. Output can be multiplexed in both space and time by using the two dimensional features of a display terminal. Input can only be multiplexed in time and this function is performed by processes called Line Handlers.

A Line Handler is associated with each Terminal connected to RIG. This association is made at the time the user first accesses RIG and continues until the user leaves the system. The Line Handler allows user processes (such as Editors, Compilers, and Executives) to each have a logical input device, called a Line, which may from time to time be connected to the user's Keyboard.

The Line Handler is the first process to interpret characters typed by the user. It fields the characters which abort and suspend processes, block output, switch Virtual Terminals, etc. Such control functions are related to particular keys via a lookup table, allowing different character codes to be used with different types of physical Keyboards (see Section 6.1). This also permits the user to specify control functions himself. Appendix B contains a list of RIG control functions available to the user.

The actions associated with control characters may be circumvented by including them in a set of break-characters (see Section 3.2). Alternatively, any character prefaced with a PASS character will be treated as a 'normal' character to be queued for the line in question.

3.1 Lines

A Line is the input component of a Virtual Terminal. Any number of Lines may be created in the course of a RIG session but only one Line may be 'active', i.e., receiving characters from the user's Keyboard, at a time. Each Line has associated with it a queue of characters which were typed while the Line was active but which have not yet been requested by the process owning the Line. Whenever a Virtual Terminal's Line is active a cursor appears in that Window to distinguish it from all others.

There are two ways in which a Line may become active:

1. The process owning the currently active Line may activate another. The Editor, for example, alternately activates its command and editing Virtual Terminals, which activates the associated Lines.

----

* Some terminals require the cursor to be used to write onto the display; in such cases the cursor always returns to rest in the current Viewport of the currently active Virtual Terminal.

2. Special function keys allow a user to shift his attention from one Virtual Terminal to another within a Region, or an Image, or from one Image to the next.

In some situations it is desirable for characters typed to one Virtual Terminal to be passed on to another. For example, a command line may be partially parsed by an Executive process to determine which user process should nandle the request. The remainder of the input should then go to that process. Accordingly, when the process owning the current active Line explicitly activates another, it may specify that its input queue be transfered to the new Line.

Characters are echoed only when they are extracted from a queue in response to a request for input. This prevents characters from appearing in the 'wrong' place on the user's Display by insuring that type-ahead always goes to the correct Virtual Terminal.

3.2 Input Modes

The process owning a Virtual Terminal may request that input be collected in one of three modes.

3.2.1 Character-at-a-Time - In character-at-a-time mode a single character is returned in response to each request. echoing is optional.

3.2.2 Page-Edit - In page-edit mode characters typed by the user are allowed to modify the contents of the Virtual Terminal until a process-specified break character is typed. A process may also specify the set of 'acceptable' characters such that any characters not in that set will be ignored. Page-edit mode is used primarily for editing files; indeed, it is simply a driver for the editing facilities of the Pad Handler (see Section 4.2).

3.2.3 Line-Edit - Line-edit mode is used primarily for processing commands. All of the intra-line editing facilities of the Pad are available (see Section 4.2). A request to line-edit is processed until a break character is typed, and a set of acceptable characters may be specified.

Entire text lines or single tokens may be line-edited. A text-line logically consists of three parts: 1) prompt; 2) previous text; and 3) current input (field). This is similar to the partitioning in TENEX, TOPS-20, and numerous other systems. For example, if a command is being typed, previously parsed fields constitute the previous text; the current field consti-

tutes the current input; and the prompt is the command prompt. Hence, the following partitioning may result (user input in upper-case):

```
        Command? COPY  OLDFILE=TEMP1  NEWFILE=

        |prompt |            text            |  input
```

The user might inquire about the current input field at any time, e.g., ask for all options for which the current input is a prefix. His inquiry will be fielded by a process external to the VTC. It is therefore possible to pass partial input back and forth to the Line Handler, editing it as necessary. It may also be possible to edit the previous text. If that text has already been parsed (by a command interpreter, for instance), the results returned from the Line Handler must indicate that such text has been changed. The calling process may then reparse the complete text.

3.3 Indirect Input

When processing commands it is often useful to specify indirect sources of input, e.g., programmable function keys, macro files, or command language programs. The processing of such input is distributed between the low-level Keyboard driver, the Terminal Input and Line Handlers, and higher-level command interpreters. We have not yet finalized the design for indirect input, but make the following observations:

3.3.1 Programmable Function Keys - Programmable function keys are straightforward as long as symbolic arguments are not allowed. The Keyboard driver (microcomputer) simply generates a continuous, uninterruptable stream of characters to the Terminal Input Handler, just as if the user had typed them. If arguments are introduced it becomes necessary for either the Input or Line Handler to 'parse' the programmed string, and replace symbolic arguments with actual Keyboard input. In any case, programmable function keys would be processed whenever they were typed, and could not be 'aborted'.

3.3.2 Indirect Files - Two types of indirect files may be of use. The first type would simulate a programmable function key by taking effective control of the Keyboard; this would be useful for executing 'transcript' files of previous sessions or initiating a particular series of tools upon logging in. The only Keyboard function which would be fielded while the indirect file is being processed is 'abort'.

The second type of indirect file would replace keyboard input only for the active Virtual Terminal. In addition to 'abort', all Keyboard functions necessary for activating new Virtual Terminals, blocking output, etc., will continue to be fielded.

In either case, the same difficulties with symbolic arguments carry over from programmable function keys.

3.3.3 Command Programs - Bona fide command programs require a command language interpreter. They should be executed in the context of a 'run' command, and are not the responsibility of VTMS.

4.0 Virtual Output

The output capabilities of a Virtual Terminal are provided by processes called Pad Handlers. Pad Handlers manage disk-based data structures called Pads, each of which represents the 'store' of a Virtual Terminal. A Pad Handler maps a Pad onto specific areas of a user's Display by issuing commands to the Terminal Output Handler (see Section 6.2).

4.1 Pads

A Pad provides:

1. The ability to hold a number of lines of text up to some maximum number set by the Pad's owner.
2. Cursors which specify the current foci of attention (for both viewing and output) in the Pad.
3. The ability to perform a number of editing commands.
4. The ability to save lines scrolled out of the Pad by storing them on disk scratch files.
5. The ability to use any file as a source for new input.
6. The capacity to insure that, should its Virtual Terminal be mapped to a Display, all changes made to the Pad are reflected in changes made to its image on the Screen.

The Pad data structure is depicted in Figure 2.

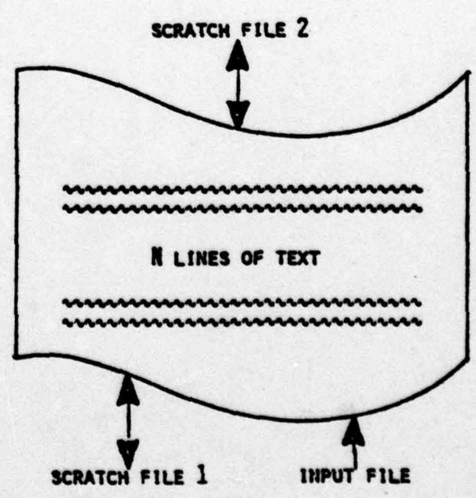


Figure 2. A Pad maintains a fixed number of text lines in memory and uses two scratch files for temporary disk storage. A file being edited will be opened as a source of additional input; only if the edit concludes normally will the file be updated. When the pad is closed normally, the scratch files are deleted; if the system crashes, the scratch files provide the means for recovering 'lost' text.

Logically, a Pad is a cursor-addressable two-dimensional right ragged array indexed by line number and character position. The maximum amount of text storable in a Pad is determined by the maximum file size on a particular RIG system. A Pad's output cursor position is updated in a fashion analogous to that of a

real terminal. Normally each character write operation increments its character index, leaving its line index fixed. An end-of-line character results in a reseting of the character index to one and an increment of the output cursor's line position.

A given Pad can be a part of more than one Virtual Terminal. This feature can be useful when a Pad contains status information updated by a single process but of interest to many system users. For example, the RIG 'banner' process makes a Pad available to all users containing the RIG system version number, date, and time. Mapping the same pad to multiple Virtual Terminals also provides the means for 'linking' or 'sharing screens.'

4.2 Editing

A range of editing features are provided by the Pad. They include:

- cursor motion by characters, words, lines, and pages
- deletion of characters, words, lines, and pages
- joining and splitting lines
- character overwrite or insertion
- string location and substitution
- text selection and transfer (mark, pick, and put)

With the exception of other VTC processes only the owner of a Pad many send messages to the associated Pad Handler asking it to perform these functions.

The contents of any RIG text file may be inserted into a Pad with a single message. In addition, some or all of a Pad may be logically copied into a named RIG file at any time. Together with the ability to select arbitrary portions of text, these features allow the Pad to be used by the RIG Editor and VTC both for the editing of files and the management of temporary text buffers.

The Pad provides all basic text editing facilities in RIG. The RIG Editor, in particular, serves mainly as a manager of multiple Pads and implements only the more complex editing commands such as definition and execution of edit macros. By placing all basic edit functions in the Pad Handler we have made them available to all Virtual Terminals.

4.3 Viewing vs. Output

The ability to modify the mapping of a Virtual Terminal to a Screen without affecting the Pad's output cursor is central to VTMS's ability to display a Virtual Terminal's past activity. Normally, movement of a Pad's output cursor also changes the mapping of Pad lines onto a Window resulting in a scrolling action, i.e., the 'newest' data is always displayed. These map-

pings may instead be defined to follow a second type of cursor, the <u>viewing</u> <u>cursor</u>. There are as many viewing cursors as there are Virtual Terminals for a given Pad, each under the control of the VTC assigned to the Virtual Terminal. These cursors are usually linked to the Pad's single output cursor in such a way that movement of one also moves the other, but the user may detach a Virtual Terminal viewing cursor for the purpose of reviewing past text and perhaps selecting it as input.

Output to a Pad may also be blocked or discarded. When blocked, the Pad refuses to process any further requests from user processes until it is unblocked. In 'autoblock' mode the Pad will automatically block when it has 'filled' the smallest Virtual Terminal with which it is associated. Special Keyboard keys control blocking, discarding, and autoblock mode.

5.0 Managing Screen Space

A multiple process environment in which simultaneous activities compete for a user's attention presents a number of problems for a screen management system:

1. A physical display may not be large enough to accommodate all the information important to all concurrent activities.

2. The user must be able to visually organize his work so that related information is arranged logically on his Screen.

3. A given process may wish to provide various viewing options (e.g., contrast, or 'optimal' sizes) without wanting to complicate its internal state with elaborate screen state information.

<u>Screen Handlers</u> solve these problems through a hierarchical decomposition of Screen space reminiscent of the way computer graphics systems divide and map data onto graphics output devices (cf. [Newman and Sproull, 1973]). The physical entities dealt with are Screens, Images, Regions, and Viewports. Logical entities are Superwindows and Windows. The mapping from logical to physical is achieved through the use of Configurations. See Figure 3.

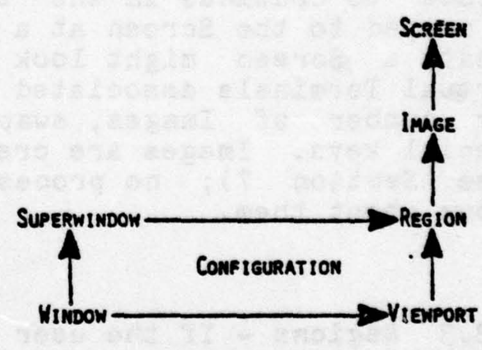


Figure 3. The dual hierarchy of Screen primitives consists of logical primitives manipulated by processes and physical primitives manipulated by the user. Logical primitives are mapped to physical ones via Configurations.

5.1 The Logical Screen

5.1.1 Windows - The <u>Window</u> is the space component of a Virtual Terminal. It represents a potential mapping of the output contained in a Pad onto a Display. Its attributes include preferred contrast (inverse, blinking, etc.) and upper and lower limits on its size when displayed.

5.1.2 Superwindows - All Windows associated with the Virtual Terminals of a particular process are collected into a logical entity called a Superwindow. This grouping ensures contiguous display of all output associated with a particular task. For example, the RIG Editor possesses four Virtual Terminals -- 'banner', 'status', 'command', and 'text'. When mapped to the Display, these Virtual Terminals should be contiguous; therefore, the four associated Windows are grouped into a Superwindow. When mapping Virtual Terminals to the Display, the user specifies Superwindows rather than Windows; a subset of the Superwindow's Windows will then be mapped (see Sections 5.3 and 7).

5.2 The Physical Screen

5.2.1 Screens - A Screen is the visible space on a Display. It has, for instance, a fixed height and width. No processes outside the user's VTC knows about the Screen unless they explicitly request the Display Profile from the Terminal Output Handler (see Section 6.2).

5.2.2 Images - Because any realizable Display is limited in size, it is desirable to have multiple screen Images, each of which may be treated as the physical Screen. This is necessary, for example, in the case where the user wishes to allocate the entire Screen to a particular process, while allowing other processes to continue in the 'background;' those processes may then be mapped to the Screen at a later time. Literally, an Image is "what a Screen might look like;" it contains a subset of those Virtual Terminals associated with the user. The user may define any number of Images, swapping between them through the use of special keys. Images are created by the user via his Monitor (see Section 7); no process outside the user's VTC and Monitor knows about them.

5.2.3 Regions - If the user wants to observe multiple processes simultaneously it may be desirable to map their output onto the Display simultaneously -- i.e., map them into the same Image. If, for example, the user wants to transfer a file across the ARPANET, monitoring its progress, while at the same time editing a local file, he may want to map the FTP and edit onto the Display simultaneously. Each process is allocated a Region of the Screen, to which the Virtual Terminals associated with that process will be mapped (as defined by the process' Superwindow). Thus an Image is composed of a set of contiguous Regions. Regions have fixed sizes and are created by the user via his Monitor (see Section 7); no process outside the user's VTC and Monitor knows about them.

5.2.4 Viewports - A Viewport is the 2-D area of a Screen within which the contents of a single Virtual Terminal are actually viewed. A Region is a collection of contiguous Viewports. The size of a Viewport is dependent upon the size of its associated Region and the bounds specified for its associated Window. For example, the Editor possesses four Virtual Terminals; when it is mapped to a Region on the Screen, each Virtual Terminal is mapped to a Viewport. No process outside the user's VTC knows about Viewports.

5.3 Configurations

It is not always desirable to observe all Virtual Terminals (and hence Windows) associated with a particular process. In the Editor, for instance, it is useful to be able to eliminate the 'command' and 'status' Windows, and deal only with a (larger) 'text' Window. This is accomplished by forming Configurations. A Configuration is a description of the way in which a subset of a process' Virtual Terminals should be displayed; it specifies the relative positions of the Windows, their relative sizes as a percentage of the whole, and actual viewing conditions. Each Window in the Configuration (termed a Configured Window) may have a size and contrast independent of the background size and contrast specified when the Virtual Terminal was created.

It is through Configurations that Virtual Terminals are actually mapped to the Screen. Configurations thus serve much the same role as a 'boxing' function in computer graphics systems. Processes may configure their Virtual Terminals in as many ways as they desire, but are not aware of which Configuration is currently 'active'. The user swaps Configurations via a special Keyboard key.

5.4 An Example

By way of further explanation, consider the following example: A user wishes to allocate half of a 25-line terminal to one task, say a Telnet, and the other half to an edit session. Through his Monitor the user creates an Image with two Regions -- one for the Superwindow associated with the Telnet and the other for the Superwindow associated with the Editor. He then asks the Monitor to swap that new Image to his Screen. The resultant display looks like Figure 12. In the top Region are two Viewports displaying respectively the banner and command Virtual Terminals of the Telnet process. The sizes of the two Viewports are determined by the actual size of the Region (in this case 10 lines) and the relative size information contained in the currently active Configuration of the Telnet Superwindow.

In the lower edit Region there are four Viewports. Because it is often advantageous to use more space for editing or entering commands, two additional Configurations for the edit Superwindow have been defined by the Editor process. These can be

seen in Figures 9 and 10. The user can switch from one Configuration to the other through the use of a special keyboard key. The Editor process is unaware of any change of Configuration, and the characteristics of the Virtual Terminals involved are not modified.

6.0 Real Terminals

Lines and Pads represent logical Keyboards and Displays, respectively. Their physical counterparts comprise a Terminal. The Line Handler communicates with the Terminal Input Handler which manages the Keyboard; Pad Handlers communicate with the Terminal Output Handler which manages the Display. The Input and Output Handlers are the process-level interface to the interrupt-level I/O handlers. There is exactly one Input and one Output Handler for each terminal in the system. The physical characteristics of the Terminal with which any RIG process is associated may be obtained via declarative Keyboard and Display Profiles.

6.1 Input

6.1.1 Keyboards - A Keyboard is considered to be any device capable of generating distinct signals in response to user input. Although the examples presented in this paper are pictures of a Delta 4000 [DDSC, 1975], we are currently designing our own Keyboards and Displays. The layout for the proposed Keyboard is given in Appendix B.

6.1.2 Keyboard Profiles - A Keyboard Profile is constructed by the Terminal Input Handler. It maps the signals generated by the Keyboard into the Virtual Terminal control functions which they represent, e.g., octal 10 is mapped to the function DELETECHARLEFT. Each character may have at most one control function; a control function may, however, be generated by more than one character. Characters not mapped are given no special interpretation by the VTC. No process other than the Terminal Input Handler should rely on a particular signal (e.g., 8-bit code) representing a particular control function; that is, signals have no pre-assigned logical function. The Keyboard Profile is provided to any RIG process upon request; it is required by the Line Handler.

6.1.3 Terminal Input Handlers - The function of a Terminal Input Handler is to collect data from a RIG Keyboard, usually in response to a request for input from a Line Handler. The Terminal Input Handler is also responsible for creating and maintaining the Keyboard Profile, and providing it to RIG processes upon request.

6.2 Output

6.2.1 Displays - A <u>Display</u> is considered to be any device on which some bounded number of text-lines may be shown simultaneously. These lines are logically numbered 1 to n, where n is the 'height' of the Display. Character positions within a line are numbered 1 to m, where m is the 'width' of the Display. A Display may, however, provide many different contrast characteristics -- color or reverse video, intensity, blinking, underlined, etc. Each characteristic may be represented by a bit in a 'contrast' mask; thus Virtual Terminals may specify any combination of characteristics, while particular Displays select those for which they are enabled.*

The format of Display commands has evolved out of a desire to both minimize the amount of state information maintained by the Terminal Output Handler, and to make them terminal-transparent (i.e., independent of any particular Terminal). The commands attempt to incorporate the features provided by most reasonable (Page Mode) Terminals, while leaving out some provided by more intelligent Terminals. The current commands are:

- alert (ring bell)
- clear screen
- clear to end of line
- delete character
- delete line
- insert character
- insert line
- move cursor
- (over)write character
- (over)write line

6.2.2 Display Profiles - A <u>Display Profile</u> is constructed by the Terminal Output Handler. It currently contains the height and width of the Display, and the contrast characteristics supported. The Display Profile is provided to any RIG process upon request; it is required by the Screen and Pad Handlers.

---

* Such Displays are commonly referred to as Page Mode Terminals. Data Entry Terminals support the concept of 'fields' and various protection attributes, such as unmodifiable, enable picks, and accept numeric input only. Date Entry Terminals may eventually be supported directly via the Terminal Output Handler; they can currently be emulated via the Pad Handler. See [Andrews, 1974] and [Irby, 1974] for some valuable comments about the requirements for an effective display device.

6.2.3 Terminal Output Handlers - The function of a Terminal Output Handler is to translate the terminal-transparent commands sent, for example, by the Screen and Pad Handlers into control signals which the particular Display can understand. The Terminal Output Handler is also responsible for creating and maintaining the Display Profile, and providing it to RIG processes upon request.

6.3 Introducing New Terminals

When a new Terminal is brought into the system, a Terminal Input Handler and Terminal Output Handler must be provided for it. The remaining components of the VTC tailor their actions, at run-time, on the basis of the resulting Keyboard and Display Profiles, and need not be changed.* Moreover, the base functions of the Input and Output Handlers do not change; all Terminal Output Handler's must, for example, handle line deletion, insertion, etc. Only the specific control signals for the terminal at hand must differ. Thus the current source code, for both the Terminal Input and Output Handler, consists of a general handler which for each terminal is combined with a terminal-specific set of support routines.

----

* This is similar to the 'negotiation of options' phase of the various Virtual Terminal Protocols employed in communications networks.

7.0 An Example

Outlining the use of VTMS requires the introduction of two new processes, the Monitor and the Executive. When the user enters RIG he is talking to the Monitor. It is responsible for managing the user's Terminal by allocating Regions to particular processes, grouping processes into Images, etc. The Monitor is also capable of showing current system status or killing errant processes.

To perform any bona fide task -- e.g., edit or file transfer -- requires an Executive. An Executive serves much the same function as a TOPS-10 Monitor or TENEX Executive (fork). Each Executive can perform at most one task at time. The user may, however, have as many Executives (and hence processes) running simultaneously as he desires. Exactly one of them may accept keyboard input at a time; it is termed the <u>active</u> Executive (and hence Virtual Terminal). At the user's request, any combination of Executives may be mapped to the Screen (in the form of an Image). The Monitor maintains state as to each Executive and can inform the user at any time what each of them is doing. Details of Monitors, Executives, and the command interface may be found in [Lantz, 1979; Lantz, in prep.].

The explanatory text for this example resides completely in the captions for Figures 4 through 17.

Figure 4. The initial Terminal state.

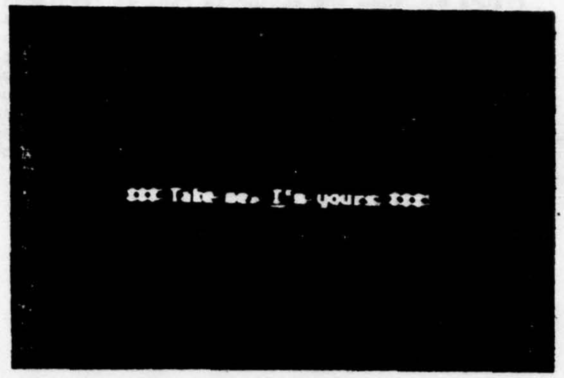


Figure 5. When the Terminal is in the state depicted in Figure 4 and the user types a character, the Screen looks like this. There are two processes mapped, the Status Server and the Monitor. The Status Server is mapped to the upper 5-line Region composed of a 1-line Viewport for the RIG banner, and a 4-line Viewport for RIG status information. The Monitor is mapped to the lower 20-line Region composed of a 1-line Viewport for the Monitor banner and a 19-line Viewport for Monitor command interaction. Hence, the Status Server and Monitor each possess two Virtual Terminals.

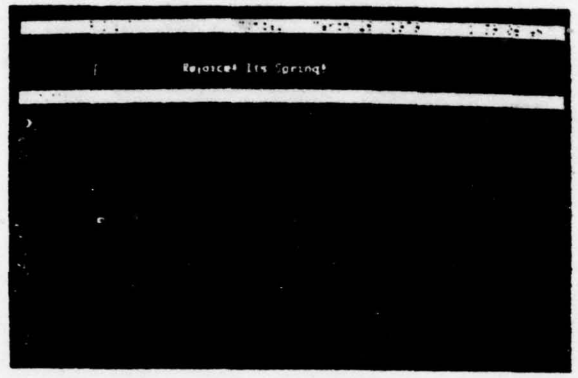


Figure 6. The available Monitor commands deal with Images, Regions, and Executives. The typical way of starting an Executive is 'SpawnExecutive'; this creates the Executive, creates an Image for it, and swaps to that Image.

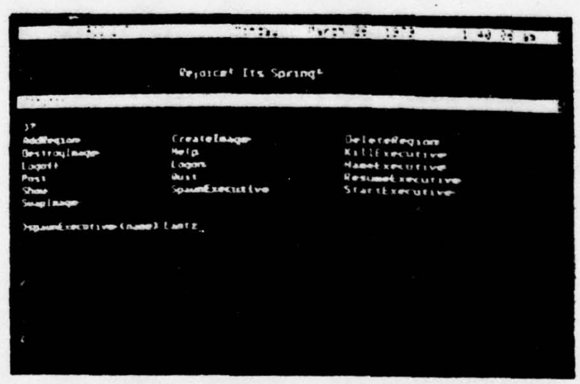


Figure 7. This Image consists of one Region to which the Executive is mapped. The Executive, like the Monitor, possesses two Virtual Terminals. The banner Virtual Terminal is mapped to the 1-line Viewport; the command interaction Virtual Terminal is mapped to the 24-line Viewport. The available Executive commands manage files, and run various tools, e.g., edit, and telnet.

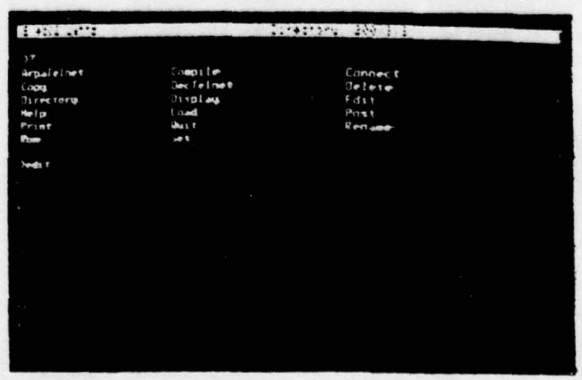


Figure 8. When the editor is run, it 'replaces' the Executive's command interaction Virtual Terminal with three of its own, one for status, one for command interaction, and one for text-editing. Together with the old banner, these four Virtual Terminals comprise a new Configuration of the Superwindow associated with the Executive. The user changes between the command and text Virtual Terminals via the CHANGEVIEWPORT key.

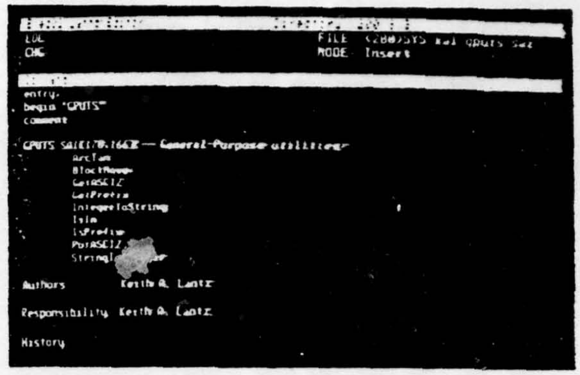


Figure 9. The default editor Configuration allows only one line for command interaction -- in order to maximize the space available for text-editing. If, however, the user needs assistance in inputing a command, he may require more lines for the command Virtual Terminal. This is provided by creating another Configuration containing only the command and text Virtual Terminals.

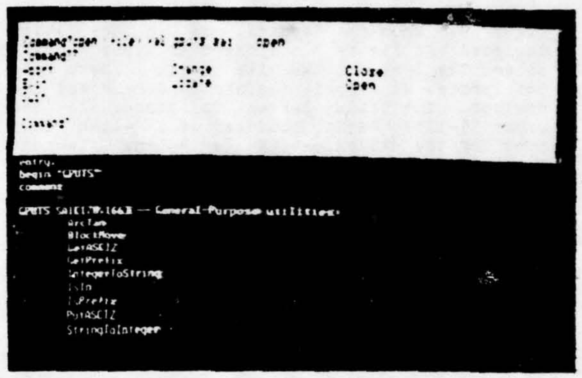


Figure 10. On the other hand, the user may want all the available editor space devoted to text-editing. Yet another Configuration composed of the single text Virtual Terminal provides this. Successive Configurations are activated via the CHANGECONFIGURATION key. In sum, the editor manages three Virtual Terminals grouped into three different Configurations; it knows nothing of how it is mapped to the Screen. When the editor 'dies', all Virtual Terminals and Configurations created by it disappear from the system, and the Executive regains control.

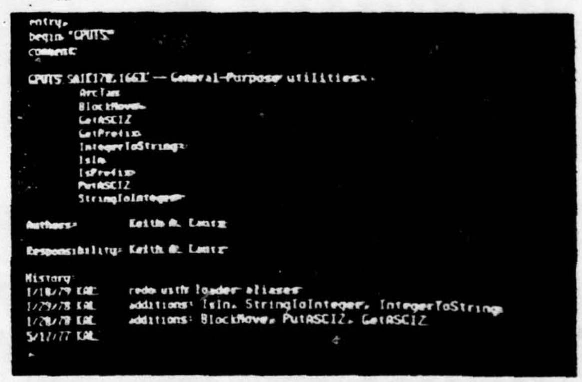


Figure 11. Assume the user has finished editing a program which runs on the PDP-10. He wants to transfer it to the PDP-10 and compile it, but can fix any errors in the local file maintained on RIG. To do so, he can map dec-telnet and the editor to the same Image. He first returns to the Monitor via the RESUMEMONITOR key. Via 'StartExecutive' he starts a new Executive from which to run dec-telnet. 'CreateImage' creates a 10-line Image containing it. The editor is added to the Image via 'AddRegion', i.e., a 15-line Region for the editor is added to the 10-line dec-telnet Region to create a 25-line Image.

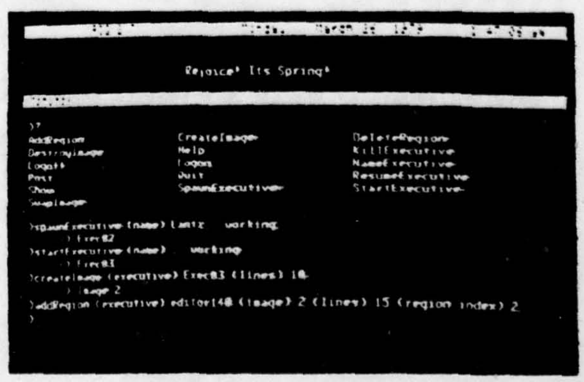


Figure 12. This Image may be activated via 'SwapImage' or the CHANGEIMAGE key. The user may change between dec-telnet and the editor via the CHANGEREGION key.

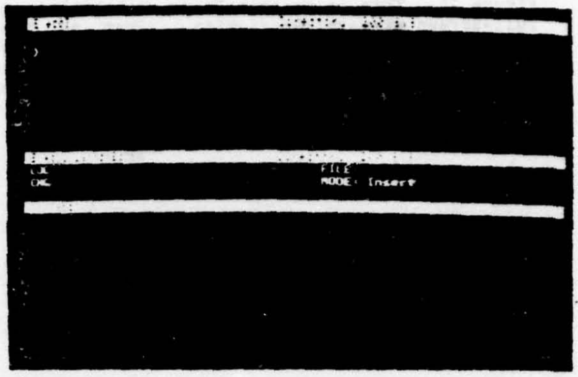


Figure 13. The user runs dec-telnet and transfers the source file to the PDP-10. When compiling, an error is found. The necessary change is made in the local file.

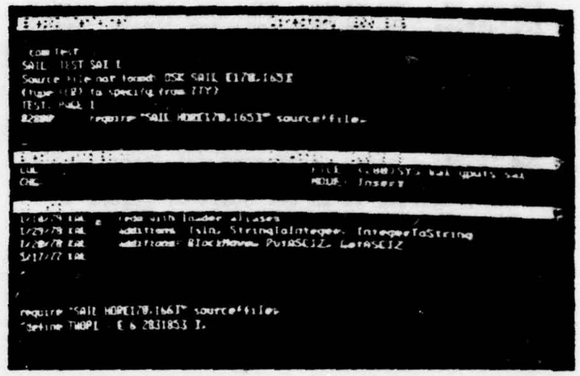


Figure 14. The compiler continues and finds another error. The user makes the necessary change.

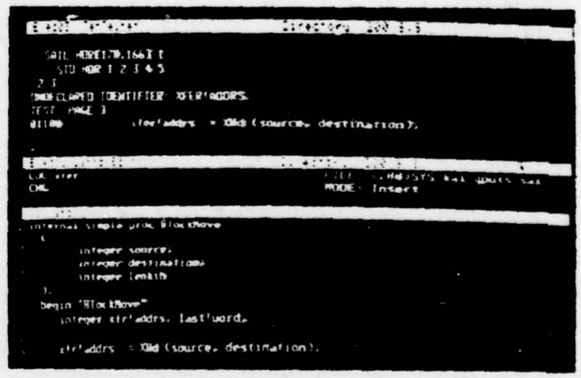


Figure 15. After the compilation is finished, the user closes the file and transfers it to the PDP-10 again. There it is recompiled without errors.

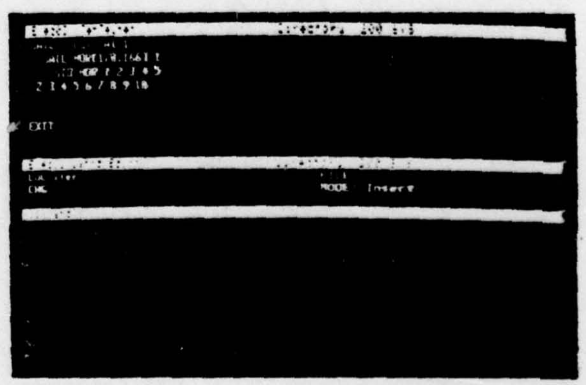


Figure 16. This is another Image containing a full-screen dec-telnet Region. Just as the editor is currently mapped to two Images, so is dec-telnet, further demonstrating the independence of the two processes from their mappings to the Screen.

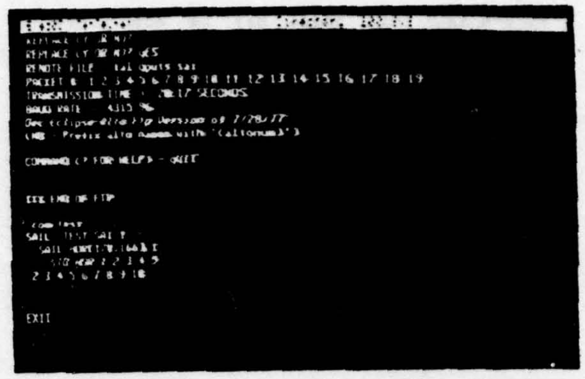
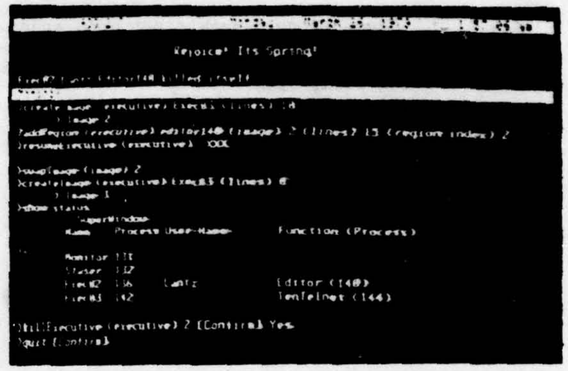


Figure 17. The status of the user's various Executives may be found via the 'Show' command. Any of the specified names for each Executive -- e.g., Exec02, Editor, or just 2 -- may be used in any commands requiring the specification of an Executive. 'KillExecutive' will kill the specified Executive and any associated tool; a death notice is posted in the Status window. The 'Quit' command terminates all Executives and any associated tools, and reinitializes the Screen, leaving it in the state of Figure 4.



8.0 Concluding Remarks

8.1 Implementation

VTMS is currently implemented on two Data General Eclipses supporting seven terminals. The terminals currently supported are the Delta Data Systems Corporation Delta 4000 [DDSC, 1975] and Research Inc. Teleray 1060 [RI, 1978]. Initially implemented for the Delta 4000, it required some 3 hours to incorporate the Teleray 1060. In both cases the entire VTC resides on the Eclipse to which the terminal is attached. Four Xerox Altos [Kay, 1977] -- 16-bit 64KW minicomputers with 606x808 raster displays -- can also be incorporated into VTMS by making them look like Delta 4000's; in this case the VTC is distributed, with the Terminal Input and Output Handlers resident on the Alto, and the remaining components on an Eclipse. All support code is written in BCPL [Richards, 1969; Curry, 1977].

Several features discussed above have not yet been implemented: the use of multiple 'viewing' cursors per Pad; text selection and placement; certain editing features such as cursor movement and deletion by word; indirect input; blocking, discarding, and autoblocking output. The technical problems encountered in each instance have, for the most part, been solved; the primary problem has been lack of manpower.

Work on VTMS began in the fall of 1975. It first came on line during the summer of 1976 as part of an experimental RIG [Ball, et al., 1976]. Experience with that system and with NEXUS, a stand-alone network access program running on the Altos [Feldman and Rashid, 1977], resulted in a number of fundamental changes in philosophy and design. The version of VTMS described here became operational in November 1978. Overall some 2-3 man years have been invested in design and implementation. Further details may be found in [Lantz and Rashid, 1979].

8.2 Future Work

User profiles will allow the user much greater control over his own display environment. It may specify a different set of control keys to match, for example, those the user uses on the PDP-10. The user profile can contain function definitions to be downloaded into programmable function keys.

Graphical I/O has been excluded from VTMS for reasons cited by [Irby, 1974]. There are, however, no fundamental inconsistencies which would prevent it. The Terminal Input Handler could be extended to allow the use of pick devices, keysets, menus, etc; the data sent to the Line Handler would be marked appropriately. The Terminal Output Handler's repertoire of commands would be extended to provide, for example, 'DRAW LINE'. Because of the difficulty of saving graphical data, however, it does not seem appropriate to include the Pad in the graphic output loop. (cf. [Caruthers, et al., 1977; Denning, 1978; GSPC, 1977; Heilman and Marchant, 1978; van den Bos, 1978; Wallace, 1976])

Graphical I/O implies that it may be useful for some tools to gain more direct control of the Terminal. For example, when talking to a remote system via a TELNET protocol it may be possible to use a full-screen editor; since the remote system does not know about Virtual Terminals, it must assume it has control of the terminal. The user can easily create an Image containing only the desired tool, but unless the Virtual Terminal Controller is circumvented, control characters will still be fielded locally. One possible solution is the introduction of 'transparent' data, i.e., data about which the VTC makes no further assumptions.

Dealing with Data Entry Terminals or other Displays which may be partitioned horizontally presents no special problems. The Screen Handler and Pad Handlers would have to know the widths of each Region. The data passed to the Terminal Output Handler would usually include two cursor positions instead of one -- the beginning and ending position in the sub-line to be displayed.

When dealing with more intelligent end-machines such as the Alto, it may be wise to offload various components of the Virtual Terminal Controller. Since each component is self-sufficient and communicates with the others solely via messages, no problems should arise. Indeed, the current implementation of the VTC for the Altos consists of Terminal Input and Output Handlers on the Altos, and the remaining components on the Eclipses. It would be much more effective, however, to implement different Screen and Pad Handlers as well; in particular, the 'sheet of paper' approach introduced in Smalltalk [LRG, 1976], extended by Teitelman in DLISP [Teitelman, 1977], and toyed with at Rochester by Ed Burke, would seem appropriate. It may also be possible to offload much of the VTC into the microcomputer installed in our future Keyboards -- a la the Line Processor for NLS [Andrews, 1974].

Further analysis with respect to communications networks is also indicated. Negotiation of options would allow application processes to tailor its own actions on the basis of what its available Terminal (and hence Virtual Terminal) could provide. The overhead incurred by distributing the VTC can be annoying to the user who has to wait for his characters to echo; this suggests the use of more local echoing such as is provided via the TELNET Remote Controlled Transmission and Echoing option. (cf. [Bauwens and Magnee, 1978; Davidson, 1977; Day, 1977; Postel and Crocker, 1977; Schicker, 1978])

8.3 Summary

VTMS addresses issues similar to those of Virtual Terminals and Virtual Terminal Protocols for communications networks (e.g., ARPANET TELNET). Device-independence is ensured by the fact that only the Virtual Terminal Controller and the user are aware of what a particular Terminal 'looks like' or how it is being used. The breakdown of the VTC into multiple processes allows it to be easily distributed.

In addition, VTMS augments the capabilities of individual Virtual Terminals and addresses the issue of managing a large number of Virtual Terminals associated with many independent processes, only some of which may be visible on a single real Terminal. The technique of using different Virtual Terminals for different tasks allows a user to suspend an operation, perform other operations, and then return without loss of context. This leads to a wealth of function not found in typical systems.

We began by presenting three laws of terminal management. In closing it is reasonable to ask how well these have been observed:

1. Law one is achieved through a flexible mapping scheme under user control. A user is able to define any number of Images containing different arrangements of Superwindows, with each Superwindow specifying a process-defined Configuration of Virtual Terminals. Input may be directed at any Virtual Terminal, and any Image may be selected for display at any time.

2. Law two is insured by providing no mechanism by which a process can determine how, when, or if a given Virtual Terminal is actually mapped to a physical device.

3. All output generated by Virtual Terminals is maintained by active data structures called Pads so that no data is ever lost by the system except that which the user explicitly throws away. This satisfies law three.

Acknowledgments

We would like to acknowledge the assistance of Edward J. Burke and Edward T. Smith in implementing some of the ideas presented herein. The efforts of the remaining RIG Working Group are also appreciated.

Many of the ideas embodied in VTMS came out of the experience gained by the Computer Science community with graphic display systems [Denning, 1978; Ewald and Fryer, 1978; GSPC, 1977; Newman and Sproull, 1973]; network virtual terminal and graphics protocols [Bauwens and Magnee, 1978; Davidson, 1977; Day, 1977; Maleson, Nabielsky, and Rashid, 1976; Postel and Crocker, 1977; Schicker and Duenki, 1978; Sproull and Thomas, 1974]; and multiple process terminal control [Swinehart, 1974; LRG, 1976; Teitelman, 1977; Rothenburg, 1977]. NLS/Augment must be singled out as having easily stood the test of time, and for giving user interfaces a good name before there were effective systems to interface the user to [Andrews, 1974; Engelbart, 1968; Engelbart, 1976; Irby, 1974; Seybold, 1978; Watson, 1976]. Systems which did not directly influence VTMS but present some similar ideas include the TSO Job Session Manager [McCrossin, O'Hara, and Koster, 1978], ZONES [Luca, 1975], POCCNET [des Jardins and Swanson, 1976], and TTDL [Krebs, Bumgardner, and Northwood, 1976].

References

Andrews, D.I., "Line processor -- a device for amplification of display terminal capabilities for text manipulation", Proc. AFIPS NCC, 43, June 1974, Chicago, Il., 257-265.

Ball, J.E., J.A. Feldman, J.R. Low, R.F. Rashid, P.D. Rovner, "RIG, Rochester's Intelligent Gateway: System Overview", IEEE Transactions on Software Engineering, SE-2, 4, December 1976, 321-328. Also available as TR5, Computer Science Department, University of Rochester.

Ball, J.E., et al., "Experiences with RIG", TR43, Computer Science Department, University of Rochester, in preparation.

Bauwens, E. and F. Magnee, "The Virtual Terminal Approach in the Belgian University Network", Computer Networks, 2, 4/5, September/October 1978, 297-311.

Caruthers, L.C., J. van den Bos, and A. van Dam, "GPGS: A Device-independent General Purpose Graphic System for Stand-alone and Satellite Graphics", Computer Graphics, 11, 2, Summer 1977, 112-119. (SIGGRAPH '77 Proceedings, San Jose, Ca.)

Curry, J.E., et al., BCPL Reference Manual, Computer Sciences Laboratory, Xerox Palo Alto Research Center, May 1977.

Davidson, J., W. Hathaway, J. Postel, N. Mimno, R. Thomas, and D. Walden, "The ARPANET TELNET Protocol: Its Purpose, Principles, Implementation, and Impact on Host Operating System Design", Proc. 5th Data Communications Symposium, Snowbird, Utah, September 1977, 4-10 - 4-18.

Day, J., "TELNET Data Entry Terminal Option", RFC 732, NIC 41762, September 1977.

Delta Data Systems Corporation, "Operators Manual for Delta 4000 Family of Video Display Terminals", June 1975.

Denning, P. (Ed.), Computing Surveys, 10, 4, December 1978. (Special Issue: Graphics Standards)

des Jardins, R. and T. Swanson, "Virtual Terminals for Spacecraft Payload Operations", Proc. ACM/IEEE Symposium on Computer Networks: Trends and Applications, Gaithersburg, Md., November 1976, 11-18.

Engelbart, D.C., and W.K. English, "A research center for augmenting human intellect", Proc. AFIPS FJCC, 33, San Francisco, Ca., 1968, 395-410.

Engelbart, D.C., "Knowledge Workshop Development", Final Report, SRI Project 1868, Augmentation Research Center, SRI International, January 1976.

Ewald, R.H. and R. Fryer (Eds.), "Final report of the GSPC State-of-the-Art Subcommittee", Computer Graphics, 12, 1-2, June 1978, 14-169.

Feldman, J.A. and R.F. Rashid, "System Support for a Distributed Image Understanding Program", Proc. DARPA Image Understanding Workshop, Minneapolis, Mn., April 1977, 83-86.

Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH", Computer Graphics, 11, 3, Fall 1977. (SIGGRAPH '77 Proceedings, San Jose, Ca.)

Heilman, R.L., and J.M. Marchant, "TIGS: An overview of the Terminal Independent Graphics System", Computer Graphics, 12, 3, August 1978, 293-297. (SIGGRAPH '78 Proceedings, Atlanta, Ga.)

Irby, C.H., "Display techniques for interactive text manipulation", Proc. AFIPS NCC, 43, Chicago, Il., June 1974, 247-255.

Kay, A.C., "Microelectronics and the Personal Computer", Scientific American, 237, 3, September 1977, 231-244.

Krebs, C.E., C. Bumgardner, and T. Northwood, "Terminal transparent display language (TTDL)," Proc. AFIPS NCC, 45, New York, NY, June 1976, 365-371.

Lantz, K.A., "RIG User's Guide: System Version 2.5", Computer Science Department, University of Rochester, February 1979.

Lantz, K.A., and R.F. Rashid, "RIG Reference Manual: Virtual Terminal Control", Internal Memo, Computer Science Department, University of Rochester, March 1979.

Lantz, K.A., Ph.D. Thesis, Computer Science Department, University of Rochester, in preparation.

Learning Research Group, "Personal Dynamic Media", SSL 76-1, Xerox Palo Alto Research Center, 1976. Excerpts appeared in IEEE Computer, March 1977.

Luca, R., "ZONES: A Solution to the Problem of Dynamic Screen Formatting in CRT-Based Networks", Proc. 4th Data Communications Symposium, Quebec City, Canada, October 1975, 1-1 - 1-7.

Maleson, J., J. Nabielsky, and R.F. Rashid, "The Rochester Image Protocol", Internal Memo, Computer Science Department, University of Rochester, April 1976.

McCrossin, J.M., R.P. O'Hara, and L.R. Koster, "A time-sharing display terminal session manager", IBM System Journal, 17, 3, 1978, 260-275.

Newman, W.M., and Sproull, R.F. Principles of Interactive Computer Graphics. McGraw-Hill Book Company, New York, 1973.

Postel, J. and D. Crocker, "Remote Controlled Transmission and Echoing Telnet Option", RFC 726, NIC 39237, September 1977.

Research Inc., "Teleray 1000 Series Instruction Manual", 1978.

Richards, M., "BCPL: A tool for compiler writing and systems programming", Proc. AFIPS SJCC, 34, Boston, Ma., May 1969, 557-566.

Roberts, L.G. and B.D. Wessler, "The ARPA Network", Advanced Research Projects Agency, Information Processing Techniques, Washington, D.C., May 1971.

Rothenburg, J., "SIGMA Message Service Reference Manual, Version 1.6", Working Paper ISI/WP-8, USC Information Science Institute, November 1977.

Schicker, P. and A. Duenki, "The Virtual Terminal Definition", Computer Networks, 2, 6, December 1978, 429-441.

Seybold, P.B., "TYMSHARE'S AUGMENT: Heralding a New Era", The Seybold Report on Word Processing, 1, 9, October 1978, WP-1 - WP-16.

Sproull, R.F., and E.L. Thomas, "A Network Graphics Protocol", Computer Graphics, 8, 3 Fall 1974.

Swinehart, D.C., "Copilot: A Multiple Process Approach to Interactive Programming Systems", Memo AIM-230, Artificial Intelligence Laboratory, Stanford University, July 1974.

Teitelman, W., "A Display Oriented Programmer's Assistant", CSL 77-3, Xerox Palo Alto Research Center, March 1977. Excerpts appeared in Proc. 5th IJCAI, August 1977, 905-915.

van den Bos, J., "Definition and Use of Higher-level Graphics Input Tools", Computer Graphics, 12, 3, August 1978, 38-42. (SIGGRAPH '78 Proceedings, Atlanta, Ga.)

Wallace, V.L., "The Semantics of Graphic Input Devices", Computer Graphics, 10, 1, Spring 1976, 61-65. (Proc. ACM Symposium on Graphic Languages, Miami Beach, Fa.)

Watson, R.W., "User interface design issues for a large interactive system", Proc. AFIPS NCC, 45, New York, NY, June 1976, 357-364.

Appendix A: Glossary

Configuration - A collection of Configured Windows. Literally "what a Region of the Screen might look like," it describes how a Superwindow should be mapped to a Region.

Configured Window - Literally "what a Viewport of a Region might look like," it describes how a Window should be mapped to a Viewport in terms of preferred size and contrast.

Display - Any physical display device; usually a Page Mode Terminal.

Image - A collection of Regions. Literally, "what a Screen might look like."

Keyboard - Any keyboard attached to a Terminal.

Line - A logical connection to the user's Keyboard through which a process may collect input.

Pad - A data structure which emulates the text storage characteristics and editing capability of a Display with infinite memory.

Region - An instance of a Superwindow mapped to the Screen. Regions are contained within Images.

Screen - The visible space on a Display.

Superwindow - A data structure which groups all the Virtual Terminal Windows associated with a particular process, and all possible Configurations of those Virtual Terminals. When a Superwindow is displayed on a Screen it is mapped to a Region.

Terminal - The physical device characterized by a Keyboard and a Display.

Viewport - An instance of a (Configured) Window mapped to a Screen. Viewports are contained within Regions.

Virtual Terminal - A pseudo-terminal from which processes collect keyboard input and to which they direct display output. A Virtual Terminal consists of a Window, a Pad, and a Line.

Virtual Terminal Controller (VTC) - The pseudo-device which manages all Virtual Terminals associated with a particular user.

Window - The space component of a Virtual Terminal, it contains default preferred size information, contrast, etc. All Windows belonging to a particular process are grouped into a Superwindow. When a Window is displayed on a Screen it is mapped to a Viewport.

Appendix B: Keyboard and Control Functions

The Keyboard being designed will have a layout as depicted in Figure B-1.

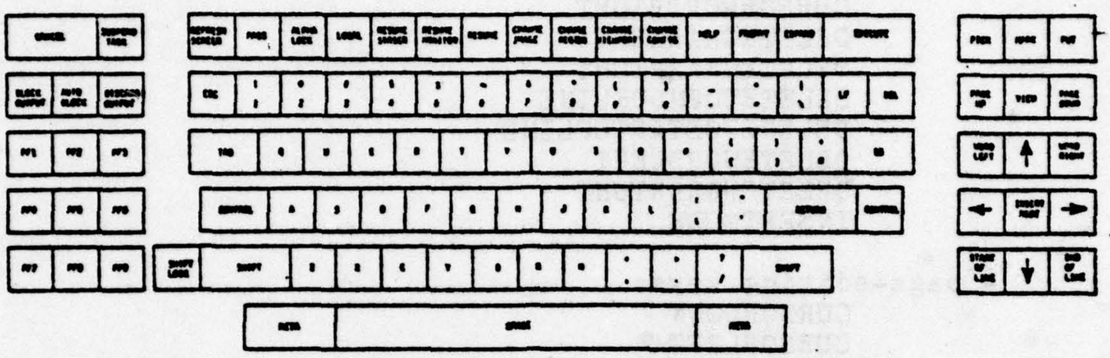


Figure B-1. The proposed Keyboard. The base keyboard, without the two control pads on either side and the row of control keys at the top, is compatible with our Alto I keyboards. It is possible to replicate any control function on the base keyboard via the CONTROL and META keys.

Four classes of control functions are of interest:

- screen and task management keys:
  ABORTTASK
  ALPHALOCK
  AUTOBLOCK
  BLOCKOUTPUT
  CHANGECONFIGURATION
  CHANGEIMAGE
  CHANGEREGION
  CHANGEVIEWPORT
  DELETEIMAGE
  DELETEMARK
  DELETEREGION
  DISCARDOUTPUT
  INDIRECTINPUT
  MARK
  PASS
  PICK
  PUT
  REFRESHSCREEN
  RESUME
  RESUMEMONITOR
  RESUMESTASER
  SUSPENDTASK
  VIEW

- line-editing keys:
  - CURSORLEFT
  - CURSORRIGHT
  - CURSORTOENDOFLINE
  - CURSORTOSTARTOFLINE
  - CURSORWORDLEFT
  - CURSORWORDRIGHT
  - DELETECHARLEFT
  - DELETECHARRIGHT
  - DELETETOENDOFLINE
  - DELETETOSTARTOFLINE
  - DELETEWORDLEFT
  - DELETEWORDRIGHT
  - INSERTMODE

- page-editing keys:
  - CURSORDOWN
  - CURSORLEFT
  - CURSORRIGHT
  - CURSORTOENDOFLINE
  - CURSORTOSTARTOFLINE
  - CURSORUP
  - CURSORWORDLEFT
  - CURSORWORDRIGHT
  - DELETECHARLEFT
  - DELETECHARRIGHT
  - DELETEPAGEDOWN
  - DELETEPAGEUP
  - DELETETOENDOFLINE
  - DELETETOSTARTOFLINE
  - DELETEWORDLEFT
  - DELETEWORDRIGHT
  - INSERTMODE
  - JOINLINE
  - PAGEDOWN
  - PAGEUP
  - SPLITLINE

- command-input keys:
  - ASSIGNSLOT
  - CANCEL
  - COMMENT
  - EXECUTE
  - EXPAND
  - HELP
  - PROMPT

In the sequel, the syntax for control functions is <function> [<default Keyboard key>].

ABORTTASK [shift-CANCEL] - abort the task associated with the active VT

ALPHALOCK [ALPHA LOCK] - software shift-lock for the active VT (toggle)

ASSIGNSLOT [=] - assign a value to a command slot

AUTOBLOCK [AUTO BLOCK] - toggle the auto-block feature for the active VT

BLOCKOUTPUT [BLOCK OUTPUT] - block/unblock output to the active VT (toggle)

CANCEL [CANCEL] - "cancel" the current input

CHANGECONFIGURATION [CHANGE CONFIG] - change active Configuration

CHANGEIMAGE [CHANGE IMAGE] - change active Image

CHANGEREGION [CHANGE REGION] - change active Region

CHANGEVIEWPORT [CHANGE VIEWPORT]- change active Viewport

COMMENT [!] - in line-edit mode, treat remainder of the line as a comment

CURSORDOWN [down arrow] - move cursor down one line

CURSORLEFT [left arrow] - move cursor left one character

CURSORRIGHT [right arrow] - move cursor right one character

CURSORTOENDOFLINE [END OF LINE] - move cursor to end of current line

CURSORTOSTARTOFLINE [START OF LINE] - move cursor to start of current line (or field)

CURSORUP [up arrow] - move cursor up one line

CURSORWORDLEFT [WORD LEFT] - move cursor left one 'word'

CURSORWORDRIGHT [WORD RIGHT] - move cursor right one 'word'

DELETECHARLEFT [BS or shift-left arrow] - 'backspace' one character

DELETECHARRIGHT [shift-right arrow] - delete the character at the cursor

DELETEIMAGE [shift-CHANGE IMAGE] - delete the active Image

DELETEMARK [shift-MARK] - delete a 'mark'

DELETEPAGEDOWN [shift-PAGE DOWN] - delete a 'page' of text downwards

DELETEPAGEUP [shift-PAGE UP] - delete a 'page' of text upwards

DELETEREGION [shift-CHANGE REGION] - delete the active Region

DELETETOENDOFLINE [shift-END OF LINE] - delete to the end of the current line

DELETETOSTARTOFLINE [shift-START OF LINE] - delete to the start of the current line or field

DELETEWORDLEFT [DEL or shift-WORD LEFT] - delete one 'word' to the left

DELETEWORDRIGHT [shift-WORD RIGHT] - delete one 'word' to the right

DISCARDOUTPUT [DISCARD OUTPUT] - discard/don't-discard output to the active VT (toggle)

EXECUTE [EXECUTE] - "execute" a command; or confirm; or answer "yes"

EXPAND [EXPAND] - "expand" the current input

HELP [HELP] - display some tutorial help related to the current input

INDIRECTINPUT [@] - take input *from* a file

INSERTMODE [INSERT MODE] - toggle 'insert' mode

JOINLINE [shift-up arrow] - DELETETOENDOFLINE, then append following line

LOCAL [LOCAL] - go into 'local' mode in order to communicate directly with the Terminal

MARK [MARK] - place a 'mark' in a pad (for pick and put)

PAGEDOWN [PAGE DOWN] - scroll a 'page' downwards

PAGEUP [PAGE UP] - scroll a 'page' upwards

PASS [PASS] - pass the next character as a 'normal' character; i.e., don't interpret it as a control function

PICK [PICK] - select the 'marked' text

PROMPT [PROMPT] - display options, etc., related to current input

PUT [PUT] - place the last selected (picked) text into the Pad

REFRESHSCREEN [REFRESH SCREEN] - refresh the screen

RESUME [RESUME] - resume the last active Virtual Terminal

RESUMEMONITOR [RESUME MONITOR] - resume an instance of the Monitor

RESUMESTASER [RESUME STASER] - resume an instance of the Staser

SPLITLINE [shift-down arrow] - split the current line, appending it to the following line

SUSPENDTASK [SUSPEND TASK] - suspend/resume the task associated with the active VT (toggle)

VIEW [VIEW] - detach/attach the viewing cursor from the output cursor (toggle)